QROWN - Because Big Data Integration is Humanly Possible

Innovation action

D4.2 – Data acquisition framework

<table>
<thead>
<tr>
<th>Author/s</th>
<th>Tomás Pariente, Nines Sanguino, Javier Villazán, Ricardo Ruiz (ATOS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reviewed by</td>
<td>Daniel Hladky (AI4BD)</td>
</tr>
<tr>
<td>Due date</td>
<td>30.11.2018 (Updated on 30.11.2019)</td>
</tr>
<tr>
<td>Version</td>
<td>2.0</td>
</tr>
<tr>
<td>Dissemination level</td>
<td>PU</td>
</tr>
<tr>
<td>Status</td>
<td>Final</td>
</tr>
<tr>
<td>Changes</td>
<td>Changes made to the document are explained in section 1.2.</td>
</tr>
</tbody>
</table>
# Table of contents

**ABSTRACT**

**EXECUTIVE SUMMARY**

**INTRODUCTION**

Overview and structure of the Document

Tracked changes

**GENERAL DATA ACQUISITION FRAMEWORK**

Data acquisition and integration in context

Data acquisition in QROWD

Intended users

Technology stack

- NIFI
- OASC
- Context Broker
- CKAN

**STATIC DATA ACQUISITION**

Description of data

Data flow

CKAN General considerations

- CKAN Structure: Datasets, resources, metadata and naming conventions
- Versioning
- Ownership

Core static data acquisition functionality

- CSDA-1: Upload/Update a dataset into CKAN (from a URL)
  - Implementation with NiFi
- CSDA-2: Upload/Update a dataset into CKAN (from a URL + decompress)
  - Implementation with NiFi
- CSDA-3: JSON-based transformations + Update a dataset into CKAN (without versioning)
  - Implementation with NiFi
- CSDA-4: External-script transformations + Update a dataset into CKAN (without versioning)
  - Implementation with NiFi

Guided procedures

General recommendations

- CSDA-1: Upload/Update a dataset into CKAN (from a URL)
Data acquisition framework

CSDA-2: Upload/Update a dataset into CKAN (from a URL + decompress) 31
CSDA-3: JSON-based transformations + Update a dataset into CKAN 33
CSDA-4: External-script transformations + Update a dataset into CKAN 36
Application of static core templates: Integrated acquisition and FIWARE/GeoJSON transformation of Municipality of Trento datasets 39
Deployment and release of the framework
Deployment 42
Release 42
Static Requirements Validation 42

DYNAMIC DATA ACQUISITION 45
Datasets 45
Data flow 45
Context Broker general considerations 46
Core dynamic data acquisition functionality 47
CDDA-5: JSON based transformation + Update to Context broker 47
Guided Procedure 50
Deployment of the framework
Deployment 51
Release 51
Dynamic Requirements Validation 52

CONCLUSIONS 55
REFERENCES 56
ANNEX 1: Deployed dataflows 57
1030 58
1031 58
**ABSTRACT**

This document presents the main building blocks of the QROWD Data Acquisition Framework. The framework is based in the definition of several data flows created by combination of Apache NiFi templates generated in the scope of QROWD. The document presents the way new datasets can be acquired in CKAN (static datasets), or uploaded to the Orion Context Broker (dynamic or streaming data) for further use.
EXECUTIVE SUMMARY

This document is the second deliverable of QROWD WP4 and presents the main building blocks of the QROWD Data Acquisition Framework for static and dynamic datasets. It is intended mainly for developers of data-enabled applications that would like to make use of a data portal (CKAN as selected implementation) and a broker (the FIWARE Orion Context Broker as selected implementation) as data repositories. The reported framework is in context of the QROWD high-level architecture described in D8.1.

The document explains the rationale behind the data acquisition framework. The framework is based in the definition of data flows created by combination of Apache NiFi templates generated in the scope of QROWD. The templates are useful for different purposes, ranging from uploading new datasets to CKAN (static datasets), or gathering the latest status of the dynamic of streaming data to the Orion Context Broker (dynamic or streaming data). The combination of those templates allow developers to easily implement new data flows by using the user interface providing by NiFi with none or minimal programming effort.

The deliverable also explains the methodology and best practices to create data flows and perform simple data transformations in NiFi. The document shows how this is done to generate datasets in multiple formats and flavours and upload them to CKAN.

The document provides an overview about how versioning of datasets is handle in CKAN for the purpose of QROWD.

This document is accompanied with software related to the data acquisition framework (NiFi templates and processes) as well as several instantiations that allow the acquisition of many of the datasets listed in the deliverable D4.1 (Data Catalog) necessary for the WP2 Trento pilot. It also provides input to QROWD developers, especially from WP3 and WP6 to upload data to CKAN and fulfil use cases required by the pilots.
1. INTRODUCTION

1.1. Overview and structure of the document

This document is the second deliverable of WP4. It is reporting about the Data Acquisition Framework developed in the scope of QROWD. In particular, the document reports on the results of task 4.2 (Data acquisition), and as such is providing the methods and software artefacts able to fulfill the requirements for data acquisition gathered from the pilot use cases (WP1 and WP2). This involves the development and deployment of the data acquisition functionalities specified originally in deliverable D4.1. [D4.1] and polished after several iterations with the project partners.

The document is structured as follows:
- Section 1 introduces the document.
- Section 2 explains the general data acquisition framework
- Section 3 reports on the acquisition of static datasets
- Section 4 introduces the acquisition of dynamic datasets
- Section 6 summarises and concludes the document

1.2. Tracked changes

This document has been revised in November 2019 to keep updated the catalog with the latest information available in relation to datasets by the end of the project. Besides cosmetic changes, the main updates are the following:
- Add and update the new NiFi pipelines in "ANNEX 1: Deployed dataflows"
2. GENERAL DATA ACQUISITION FRAMEWORK

2.1. Data acquisition and integration in context

This section provides the context of the data acquisition framework in relation to the QROAD architecture.

Figure 1: QROAD general architecture

Figure 1 shows the QROAD architectural diagram as described in the QROAD deliverable D8.1. The Data Acquisition Framework is providing support to gather static and dynamic datasets from the different pilots and store that information either in CKAN [CKAN] (static) or in the Context Broker [FIWARE Context Broker] (dynamic) repositories.

2.2. Data acquisition in QROAD
Data acquisition framework

This deliverable presents the data acquisition framework for the collection of QROWD-relevant data. According to the definition given in deliverable D4.1, the acquisition framework should allow the extraction of data from distributed and heterogeneous sources and make it available for further usage in the project. A fully identification and assessment of the datasets to be acquired by QROWD was done in D4.1, and more specifically in the so-called Data Catalog, which was the result of D4.1: QROWD Live Data Catalog (LDC).

However, this document describes just the main data acquisition flows in QROWD. These flows are related to the main static data collection, described in section 3 of this document, and to the dynamic data collection, described in Section 4. Other QROWD work packages are collecting external data from their own internal purposes, such as:

- Real-time or mobile phone sensor data acquisition, described in D2.4 “iLOG”.
- News/Social Media streaming acquisition mechanisms, fully described in D4.4 “Crowdsourced multilingual data harvesting and extraction framework”.

Figure 2 shows a general view of the acquisition framework in the context of static and dynamic data acquisition. As depicted in the figure, other data flows such as coming form city sensors might be supported by the framework with the inclusion of FIWARE IoT Agents.

![Figure 2: Main QROWD acquisition processes explained in the document](image)

For the static and dynamic data acquisition framework presented in this deliverable, following mechanisms are put available to assist the QROWD data value chain:

- Set of acquisition and transformation components based on NiFi [Apache NiFi], CKAN [CKAN], and FIWARE Context Broker [FIWARE Context}
Data acquisition framework

- **Broker** mature open-sources technologies (See section 2.4.1, 2.4.2.2 and 2.4.2.3 respectively for more details),
- A set of guidelines for their usage and parameterization.

In the context of integration, a set of components which allow data format transformation are also included as part of the framework. See Section 3.4 for more details.

### 2.3. Intended users

The presented framework will provide users and processes data acquisition mechanisms for collecting data of different nature (static and dynamic/streaming) and from different sources (i.e. external services and repositories to QROWD or internal processes) and make them available to other QROWD processes and other actors.

In particular, the static data framework aims to provide users and processes acquisition facilities such as:

- Uploading new datasets to the central data repository, CKAN. These datasets may come from different sources of information:
  - Dataset coming from the Municipality of Trento services
  - VCE tool (See D3.2 Crowdsourcing services).
  - QROWD Fusion and Interlinking process from QROWD WP5
  - Available datasets such as: OpenStreetMap
  - Other sources
- Uploading new versions of existing datasets (versioning/backup)
- Transforming and uploading different formats of existing datasets. Some of the formats managed in QROWD are:
  - FIWARE format for data integration/homogenization purposes. FIWARE data models\(^1\) [FIWARE Data Models] are a set of harmonized data model for smart cities applications.
  - GeoJSON [H. Butler et al, 2016] for visualization purposes

On the other hand, the dynamic data acquisition framework uses the Orion Context Broker (FIWARE) to persist contextual information about the state of several assets of a city (i.e. parking lots). Consequently, the dynamic acquisition framework is intended for processes with the need of receiving information about the real time status of the city and do something with it, such as the visualization of real-time data in the Municipality and Citizen dashboards.

### 2.4. Technology stack

#### 2.4.1. **NIFI**

---

1 https://www.fiware.org/developers/data-models/
2 https://www.w3.org/RDF/
Apache NiFi is a data platform developed by the Apache Software Foundation created to move data between different systems both in real time and scheduled manner with a really intuitive and easy to use graphic interface. It was designed to tackle some of the most relevant issues in the industry when integrating data over different systems.

NiFi’s base unit of work is called a *FlowFile*. A *FlowFile* is an encapsulation of the data that will be processed with some extra attributes that define that data (like filename, last update time…). These *FlowFiles* are modified and routed by using processors.

This way of handling the data gives NiFi, out of the box, some useful features that include:

- It is highly scalable, can be clustered and easily scaled horizontally so each flowfile can be processed in a different system.\(^3\)
- It allows to use *back pressure*\(^4\) in the system. The amount of data that is stored because the target system is not able to handle it is configurable and it also allows to tell the source system to hold the ingestion.
- Provenance\(^5\) of data is stored, so you can know what processors modified that data.

In a nutshell, NiFi moves the data in the *FlowFile* from processor to processor through the connections that link them. It can also be used to route the *FlowFiles* through different paths depending on attributes or in the data itself.

On top of this engine, Apache NiFi provides the possibility to create your own custom processors in case you have any functionality that cannot be covered with the standard set of processors bundled with it. This way anyone can easily implement the required interfaces and create a processor that can, for example, read or write data to a CKAN system with NiFi but integrated in the NiFi platform without the developer having to worry about coding most of the functionalities that NiFi offers because they are added seamlessly to the custom processor.

### 2.4.2. OASC

#### 2.4.2.1. Context Broker

As a part of FIWARE, Context Broker is one of the most important components of this framework. As we have seen in the D4.1 “Data Catalog”, one of the major impacts pursued by the QROWD project is the replicability/reusability of their results. FIWARE is an open source platform component that allows us to communicate with other technologies to create a better data ecosystem. This yields in an optimize data flow that improvement the use of the data.

---

3 http://nifi.apache.org/docs/nifi-docs/html/administration-guide.html#clustering  
4 https://nifi.apache.org/docs/nifi-docs/html/user-guide.html#back-pressure  
5 https://nifi.apache.org/docs/nifi-docs/html/user-guide.html#data_provenance
Context Broker is based on the implementation of the NGSI9/NGSI10 Informational model [NGSI-9/NGSI-10 information model]. “The NGSI Context Management informational Specifications are defined by the OMA(Open Mobile Alliance)”.

The basic NGSI informational model are composed of:

- **Entities.** The virtual representation of an object of the real life. The entities have an identifier and a type.
- **Attributes.** Information relative to feature of the entities, also can contains metadata.
- **Domain Attributes.** A way to create sets of elements and group the attributes with a similar logic.
- **Context elements.** “The data structure used for exchanging information about entities”.

Context Broker is used in projects where you need to develop and deploy with data, as an intermediate component to connect data producers and data consumers. The main properties of this component can be resume as follows:

i) Register elements of the context.
ii) Manage these elements, consult and update
iii) Subscribe to these elements, that allows us to receive a notification if for example the data content a change.

Context Broker is a good tool to work with a lot of different kind of data, and also are adapted to use geolocation data and IOT (Internet Of Things) data, which are a kind of data used to define what we call a “Smart City”.

### 2.4.2.2. CKAN

Several definitions of CKAN can be found in the official website\(^6\), one of them states that “CKAN is a fully-featured, mature, open source data portal and data management solution” which can explain why CKAN is broadly used in Europe as data catalog which allows data publishers sharing collection of data with general users.

In particular, It provides features such as:

- An open-source platform that can be easily adapted and extended
- An intuitive web interface to manage datasets and organizations\(^7\).
- API to allow developers interact with CKAN\(^8\)
- A rich set of metadata for datasets annotations.
- Data storage for storage of structure data\(^9\) and data previsualization\(^10\)
- Search and discovery, geospatial features...and more functionalities\(^11\)

---
\(^6\) [https://ckan.org/about/](https://ckan.org/about/)
\(^7\) [https://ckan.org/portfolio/publish-and-manage-data/](https://ckan.org/portfolio/publish-and-manage-data/)
\(^8\) [https://docs.ckan.org/en/latest/api/index.html](https://docs.ckan.org/en/latest/api/index.html)
\(^9\) [https://ckan.org/portfolio/datastore/](https://ckan.org/portfolio/datastore/)
\(^10\) [https://ckan.org/portfolio/visualization/](https://ckan.org/portfolio/visualization/)
\(^11\) [https://ckan.org/features/](https://ckan.org/features/)
In addition to that, the Open and Agile Smart City initiative (OASC)\textsuperscript{12} which aims to provide best practices for the construction of smart cities system, rely on CKAN as one of its main pillars. It propose CKAN platform as the base standard platform for publication of static file datasets\textsuperscript{12}.

The recommendation of OASC initiative added to the before commented functionalities has contribute to select CKAN as main repository for static dataset in QROWD.

In the following section it can be found more details about the particular use of CKAN in QROWD and the operative defined.

3. STATIC DATA ACQUISITION

This section presents the “Acquisition Components of Static Data” framework (ASCD), specified in D4.1 and fully implemented in D4.2.

3.1. Description of data

The framework described in following sections encompasses the acquisition of datasets of static nature.

As it was commented in D4.1, “static data, also known as data-at-rest, is data that does not or barely change after its recording” and putting it in the context of QROWD, it refers to those datasets related with the infrastructure of the city, i.e: bike racks, paid parking zones, parking for disability people, bike lanes, schools, libraries, e-car charging stations, etc.

There is, in the context of QROWD, an additional classification of datasets attending the provenance of them:

- Datasets of type “Source”, original datasets coming from internal or external data sources, i.e: datasets coming from the Municipality of trento, resulting datasets coming from VCE component, OpenStreetMaps datasets, etc.
- Datasets of type “Intermediate”, datasets resulting from internal fusion processes which make use of “source” datasets.
- Datasets of type “Final”, dataset resulting from internal validation processes which make use of “intermediate” datasets.

\textsuperscript{12} \url{http://oascities.org/wp-content/uploads/2016/02/Open-and-Agile-Smart-Cities-Background-Document-3rd-Wave.pdf}
It is worth mentioning that the existence of different datasets classified as source, intermediate or final does not affect the use of the data acquisition framework. It will be transparent, from an acquisition point of view, whether a dataset comes from the Municipality or from a fusion process. The only distinction between them will be the publisher reflected in the name of the dataset and the tag associated to the dataset: Type dataset":{QROWD_source, QROWD_fusion, QROWD_official}

### 3.2. Data flow

Figure 4 shows the architecture to acquire, update and transform static datasets in CKAN. They are the acquisition and transformation NiFi box what will be fully explained in Section 3.4.
Data acquisition framework

Based on the consideration described in Section 2.4.2.2, CKAN is used as main repository for the management and storage of static dataset in QROWD. All the acquisition and transformation processes of static data will sink into CKAN datasets.

Several acquisition processes take part in the process of uploading datasets into the QROWD CKAN: Processes that collect datasets from the Municipality of Trento, tools collecting crowdsourcing information to complete the information about Municipality infrastructure, interlinking and fusion processes, validation processes, etc.

On the other hand, additional processes will interact with CKAN to add different formats of existing datasets: FIWARE transformations to integrate datasets into a common format, GeoJSON transformations to make easier the visualization process in the dashboard, etc.

Apart from uploading a new dataset or adding new formats to a existing one, the framework also allows update datasets by means of adding a new version of an existing format in an existing dataset. In that case, and in order to avoid inconsistencies between different formats on the same dataset, the update of a format must be complemented with the update of the rest of formats in the dataset, to provide all the formats referencing same content/version. Therefore, the user should arrange not only a format update but also a set of transformations to ensure the matching between the different formats in a dataset version.

To enable this, upload and transformation processes must be linked in some cases. Section 3.6 details an example of the integrated dataset acquisition and ulterior transformations to FIWARE and /GeoJSON formats of some datasets from the Municipality of Trento. This complex dataflow ensures the automatic triggering of all the transformations needed when a new version of a dataset is uploaded. In the example shown in Figure 18, when a new version of the impianti-sportivi dataset is uploaded, not only the resource containing the raw information is updated, but also the FIWARE and GeoJSON format are automatically update.

All the acquisition and transformations will be made through a set of NiFi-based dataflows which make easier the integration with further components in the QROWD general architecture.

3.3. CKAN General considerations

CKAN Structure: Datasets, resources, metadata and naming conventions

In QROWD, the CKAN repository will make use of following CKAN entities: Datasets, resources, and metadata. Datasets entities will be used for the abstract concept of a dataset, resources for the physical manifestations of a dataset in several formats, and a set of basic metadata for datasets annotation.

Specifically, and focusing on a particular dataset, there will be a dataset containing always the last version of a particular collection of data in CKAN. From now on this dataset will be referred as “fresh” dataset. This “fresh” dataset will have the name
Data acquisition framework

DatasetID = {DatasetName} + {Publisher}. For instance, for the dataset providing bike-racks data published by the municipality, there will be a dataset into CKAN with id “Bike_Racks_MT”.

A dataset might have several formats for the same data, therefore a CKAN dataset will have such a number of resource entries as formats availables for this particular dataset. The naming convention will be: {DatasetID} + {} + {FormatName} + {.ext}. For instance, if we have a “FIWARE” version and a “Dashboard/GeoJSON” version for the Bikes_MT dataset, the data acquisition framework will allow users have a dataset in CKAN with name “Bike_Racks_MT” with two associated resources: “Bike_Racks_MT_FIWARE.json” and “Bike_Racks_MT_GeoJSON.json”.

Finally, for the annotations associated to a dataset, there will be a metadata file, in the form of a new resource, attached to the dataset with name: {DatasetID} + {} + {Metadata}+ {.ext}. Not all datasets will have a metadata file associated. In addition to this metadata file, all datasets will be annotated with a set of basic metadata that will be stored in the metadata fields of the dataset. These basic metadata are:

- Owner/Organization
- Visibility
- tag1: “Project: String”. By default “QROWD”
- tag2: “Type versioning”:{QROWD_lastVersion, QROWD_historical}
- tag3: “Type dataset”:{QROWD_source, QROWD_fusion, QROWD_official}

Owner, Visibility, tag1, tag2, tag3 are mandatory.
Versioning

The acquisition framework will provide a “versioning” mechanism. The user will be able to manage different versions of a particular dataset, that is, when the content of a dataset is modified and the user wants to store the new version without missing old versions, the framework will automatically manage the update of a dataset backing up the previous version. Therefore, when a new version of an existing dataset arrives into the system:

1. A new dataset is created to backup all the existing formats (resources) of the “fresh” dataset. The backup dataset will have the same name as the “fresh” dataset DatasetID + “timestamp”. It could be considered as a snapshot (a copy of the state) of the “fresh” dataset at a particular point in time.
2. The “fresh” dataset (remember it contains always the last version) will be overridden with the new version. In order to do that, the resource that matches the format of the new version will be replaced by the new version and the rest of outdated resources will be removed from the “fresh” dataset.
3. If any format should be recreated based on the new version, a notification should trigger to other components in charge of creating new formats to complete the “fresh” and last version of the dataset. The notification could be
Data acquisition framework

done through the Context Broker or any other mechanism managed by NiFi processors.

Figure 6: QROWN CKAN versioning

Figure 6 shows an abstract idea of how the versioning takes place: One dataset containing the last version (always the same) and N datasets for the historical versions.

A CKAN repository example of the idea depicted above can be seen in the Figure 6. Four CKAN datasets, for the same “zone-traffico-limitato” dataset, can be seen:

- “zone-traffico-limitato” containing the last version and,
- Three additional ones for storing old versions:
  - “zone-traffico-limitato20181005_124813”, a version freezed on October, 5, 2018
  - “zone-traffico-limitato20181004_171050”, a version freezed on October, 5, 2018 at different time.
  - “zone-traffico-limitato20181004_164049”, a version freezed on October 4, 2018.
Data acquisition framework

Ownership

According to CKAN documentation: “Each dataset can belong to one or more organization. And each organization controls access to its datasets”\(^{13}\), however from QROWN acquisition point of view, each dataset belongs to one single organization, the content creator. Therefore it is possible to find two datasets containing same information (i.e: bike racks in Trento) but with different creator (assuming therefore difference content) which result in two different datasets in CKAN.

As the name of a CKAN dataset usually reflects the information contained within it and we use the name as unambiguously identifier, this normative suggests adding the creator name as part of the dataset name.

---

\(^{13}\) https://docs.ckan.org/en/2.8/maintaining/authorization.html
3.4. Core static data acquisition functionality

This section describes the NiFi core templates that implement the main functionalities offered by the static data acquisition framework:

- **CSDA-1: Upload/Update a dataset into CKAN**, a generic data flow in charge of taking a dataset (file) from a remote URL path and updating a CKAN repository. The process contemplates the “versioning” for a dataset:
  - uploading it as a new dataset (if the datasets does not exist) or
  - updating a existing CKAN dataset and store the former content as historical dataset.

- **CSDA-2: Upload/Update a dataset into CKAN (with decompression)**, same data flow than before but allowing the user download a dataset in the form of a data compressed packaged and extract/filter files to be uploaded.

- **CSDA-3: JSON-based transformations/versions of a dataset**, by mean of this functionality the user will be able to download a resource from a remote URL, perform a set of transformations over the dataset and upload the result as a new format of a existing dataset. The transformations will be in terms of
  - format: the original dataset will be transformed into JSON format;
  - structure: after the JSON transformation the user will have the possibility of performing transformations in the JSON structure, that is: change the name of the fields, create new JSON objects, or delete existing ones.
Data acquisition framework

- **CSDA-4: External-script transformation/format of a dataset**, offers the user the possibility of downloading a file from a remote URL, performing custom transformations and uploading it as a new format for a particular dataset.

These core templates can be used in two different ways:

1. **Isolated templates for basic functionalities**. For instance to upload a dataset, or a create and upload a new format of an existing dataset. Running these functionalities would be just a matter of downloading the templates and configuring them following the guidelines proposed in Section 3.5.

2. **Combination of basic templates to create complex topologies** for more advanced functionalities. For instance to create a chain of transformations and upload the resulting datasets or formats into CKAN by linking core templates, as it is described in Section 3.6. To run these complex functionalities, users should have some NiFi knowledge to establish, for instance, the needed connections between the dataflows and processors.

Finally highlighting that CSDA flows are based on standards NiFi processors and specific new NiFi processors developed in the context of QROWD. For each processors will be indicated whether is standard or custom processors.

### 3.4.1. **CSDA-1: Upload/Update a dataset into CKAN (from a URL)**

One of the main functionalities of the acquisition framework is to provide users with the possibility of automatically uploading/updating a dataset into CKAN including versioning support. It takes the dataset, in the form of a digital file format, from a remote URL, and upload it into a new dataset in case the dataset does not exist previously in CKAN, or update an existing dataset in case it is an existing one.

According to the versioning control mechanism described in Section 3.3, updating a existing dataset will imply creating a backup of the “fresh” dataset and substituting the content of it with more recent information.

Examples of uses are:
- partners acquiring datasets from the municipality of Trento,
- process generating new datasets from crowdsourcing services,
- partners acquiring datasets from OpenStreetMap

**Implementation with NiFi**

Figure 8 shows different NiFi processors used in the dataflow.
Data acquisition framework

![Diagram of QROWD CSDA-1 NiFi processors](image)

**Figure 8: QROWD CSDA-1 NiFi processors**

- “InvokeHTTP” (standard), processor in charge of downloading a file from a remote URL.
- “Define_Package_Name” (standard), processor in charge of adding custom attributes to the data flow. In particular, this processor will add:
  - an attribute to store the name of the dataset in CKAN and,
  - an attribute to give a name to the flow file. This name will be used as the name of the resource (or file) to be added to CKAN.
- “CKAN_Package_Backup” (custom), processor in charge of backing up a CKAN dataset. The backup will have the same name as the copied dataset + timestamp (backup time). The copied dataset remains as it is.
- “CKAN_Flowfile_Uploader” (custom), processor in charge of uploading a new resource (file) into a CKAN dataset.

### 3.4.2. CSDA-2: Upload/Update a dataset into CKAN (from a URL + decompress)

This process is almost the same as before with the additional functionality of decompressing the downloaded file and filtering the extracted files to determine which of them will go through the uploading process. The previous process allows downloading exactly one file which will go through the process of uploading into CKAN. This particular process will allow the user to download a file/package, decompress the package and apply a filtering step to those files the user is interested in. The resulting files from this decompress/filtering step will go through the uploading process.

Examples of use:
Data acquisition framework

- Most of the datasets downloaded from the Municipality of Trento and used in the QROWD dashboard.

Implementation with NiFi

Figure 9 shows different NiFi processors that take place in the dataflow.

This data flow includes one more processor to the NiFi processors described in the previous section:

- “UnpackContent” (standard), a processor in charge of extracting files from a compressed package and filtering the desired files that will be incorporated to the workflow.

3.4.3. CSDA-3: JSON-based transformations + Update a dataset into CKAN (without versioning)
Data acquisition framework

This dataflow offers the user very basic functionality of typical JSON-based DTL (download, transform and load) process. It will allow to download a JSON format dataset from a URL, to transform the original format into GeoJSON format, to perform changes in the JSON structure, and to upload the resulting dataset into CKAN.

Since this dataflow does not generate backup of any dataset, it produces new formats of existing versions and no new versions, it’s worth mentioning that the output of the transformation (the new format) is assumed to be uploaded as a new resource (new file) into an existing dataset, in particular into the dataset from which the file to be transformed was downloaded.

Examples of uses are:
- Original datasets transformed into FIWARE data model.

Implementation with NiFi

Figure 10 shows different NiFi processors that take place in the dataflow.

![Figure 10: QROWN CSDA-3 NiFi processors](image)
Data acquisition framework

- “Invoke HTTP” (estándar), processor in charge of downloading a file from a remote URL.
- “SplitJson” (estándar), processor in charge of splitting an input JSON array into individual entities. Through a JSON path expression, the user will be able to select the elements from the array to be splitted. From now on, there will be a such number of flowfile as elements in the selected array.
- “EvaluateJsonPath” (estándar), processor in charge of putting JSON content into attributes of the flowfile. The user can create as much new attributes as different information want to store from the JSON content.
- “UpdateAttribute” (estándar), processor in charge of adding custom attributes to the data flow. In particular, in particular this process will add:
  - an attribute to store the type of entities that are flowing through the data flow
- “JoltTransformJSON” (estándar), processor in charge of allowing JSON to JSON transformations based on a Jolt specification\(^\text{14}\).
- “MergeContent” (estándar), processor in charge of joining different entities in the dataflow into single one entity.
- “UpdateAttribute” (estándar), processor in charge of adding custom attributes to the data flow. In particular, this processor will add:
  - an attribute to give a name to the flow file. This name will be used as the name of the resource (or file) to be added to CKAN.
- “CKAN_Flowfile_Uploader” (custom), processor in charge of uploading a new resource (file) into a CKAN dataset.

### 3.4.4. CSDA-4: External-script transformations + Update a dataset into CKAN (without versioning)

This template offers the possibility to download a dataset (input file) from a remote URL, perform a transformation over the dataset using external procedures and store the resulting dataset (output file) into a CKAN repository. The transformations will be command line-based executions of external script developed by the user.

As in the previous case, this dataflow does not generate a backup of any dataset. It is worth mentioning that the output of the transformation (the new format) is assumed to be uploaded as a new resource (new file) into an existing dataset, in particular into the dataset from which the file to be transformed was downloaded.

Examples of use are:
- the dashboard transformation into GeoJSON format that does not need generate new version of a dataset.

Implementation with NiFi

\(^{14}\) http://jolt-demo.appspot.com/#inception
Figure 11 shows the different processors used in the data flow and the properties that should be configured by the user for each of them:

- **“Invoke HTTP” (estandar)**, processor in charge of downloading a dataset (file) from the remote URL. The downloaded file will be the flow file.
- **“UpdateAttribute” (estandar)**, processor in charge of adding custom attributes to the data flow. In particular, this processor will add:
  - an attribute to give a name to the flow file. This name will be used as the file name in the following processor.
- **“PutFile” (estandar)**, processor in charge of writing a flow file into disk.
- **“ExecuteStreamComand” (estandar)**, processor in charge of transforming an input file into an output file by executing some external command line-based script. As a suggestion, the processor can be configured with:
  - “ogr2ogr” as command line path (script) which performs simple transformation between file formats.
  - “-f","outputFormat","outputFile","inputFile";-s_srs;EPSG:25832;-t_srs;EPSG:4326” as command arguments. The outputFile and the inputFile will be produced and picked up respectively from the directory defined in PutFile.
Data acquisition framework

- “FetchFile” (estandar), processor in charge of fetching a local file as flow file.
- “UpdateAttribute” (estandar), processor in charge of adding custom attributes to the data flow. In particular, this processor will add:
  - an attribute to give a name to the flow file. This name will be used as the name of the resource (or file) to be added to CKAN.
- “CKAN_Flowfile_Uploader” (custom), processor in charge of uploading a new resource (file) into a CKAN dataset.

3.5. Guided procedures

This section provides a set of recommendations and basic instructions to configure and create their own data flows based on the templates provided by the Static Data Acquisition Framework (CSDA).

General recommendations

- An API Key-based authorization is required for operating with CKAN API. For getting an user and API Key, please register at http://CKAN_host/user/register
- To configure a processor in NiFi double-click on it and go to the “properties” tab.

![Figure 12: NiFi processor configuration](image)
**Data acquisition framework**

**CSDA-1: Upload/Update a dataset into CKAN (from a URL)**

Figure 13 shows the NiFi GUI to add a new element to the NiFi dataflow based on previous templates. In this case the user should select “Upload_CKAN” template and below you can find the instructions to run it.

1. **Step 1:** Go to “InvokeHTTP” processor, go to “properties” tab and fulfill:
   - **Mandatory:**
     - “Remote URL”, fill with the remote URL form where to get the dataset.
   - **Others.** Apart from the mandatory properties, other properties exist in the processor and they can remain as they are or be configured by the user. Some of them are:
     - “Basic Authentication Username”, “Basic Authentication Password” can be used for downloading with required authentication.
     - In “Scheduling” tab the user can select a “Timer driven” strategy and schedule the download task (and therefore the entire workflow).

2. **Step 2:** Go to “Define_Package_Name processor”, go to “properties” tab and fulfill:
   - **Mandatory**
Data acquisition framework

- “CKAN_package_name”, the name of the dataset in CKAN. Only accept letters, hyphen, and lowercase.
- “filename”, the name of the flowfile. This name will be used by “CKAN_Flowfile_Uploader” to set the name of the resource (or file) to be added to CKAN. It is a fixed string.

3. Step 3: Got to “CKAN_Package_Backup” processor, go to “properties” tab and fulfill:
   - Mandatory
     - “CKAN Url”, URL of the CKAN repository
     - “File Api_key”, API Key for operating with CKAN API.
     - “Name of the package to backup”, Name of the dataset to be backed up. If the dataset does not exist, nothing happens. If the dataset exist, a copy of the specified dataset will be created. According to the normative defined in Section 3.3, just “fresh” datasets are subject to be backed up. It should be lowercase.
     - “Comma-separated Tag List”, the user might specify different alphanumeric tags. As it was explained in Section 3.3 at least three tags should be fulfilled to backup a new dataset:
       - One tag indicating the project name: “String”. By default and in the context of the QROWD project the user must set always: “QROWD”
       - One tag indicating the type of dataset. As we are configuring the backup processor, the user must set always: “QROWD_historical”
       - One tag indicating the type of dataset (from a use point of view). In the context of QROWD the user has to select one of the following values:{QROWD_source, QROWD_fusion, QROWD_official}.
       - Example: “QROWD, QROWD_historical, QROWD_source”.

4. Step 4: Go to “CKAN_Flowfile_Uploader” processor, go to “properties” tab and fulfill:
   - Mandatory
     - “CKAN Url”, URL of the CKAN repository
     - “File Api_key”, API Key for operating with CKAN API.
     - “Organization id”, the creator/owner of the dataset. This property is always needed, the first time to add the organization to a dataset, and following times to check if the organization passed corresponds to the ownership of dataset in CKAN. It should be lowercase.
     - “Name_of_the_package”, the CKAN dataset that will host the new version. According to the normative defined in Section 3.3, just “fresh” datasets are subject to keep the last version. It should be lowercase.
     - “Package visibility”, the accessibility of the dataset: “public”, if anyone can access to the dataset without restrictions, or “private” restricted to some users.
     - “Comma-separated Tag List”, here the user might specify different alphanumeric tags. As it was explained in Section 3.3 at least three tags should be fulfilled to upload a new dataset:
Data acquisition framework

- One tag indicating the project name: “String”. By default and in the context of the QROWD project the user must set always: “QROWD”.
- One tag indicating the type of dataset. As we are configuring the uploader processor, the user must set: “QROWD_lastVersion”
- One tag indicating the type of dataset (from a use point of view). In the context of QROWD the user has to select one of the following values: {QROWD_source, QROWD_fusion, QROWD_official}
- Example: “QROWD, QROWD_lastVersion, QROWD_fusion”.

**CSDA-2: Upload/Update a dataset into CKAN (from a URL + decompress)**

Figure 14 shows the NiFi GUI to add a new element to the NiFi dataflow based on previous templates. In this case the user should select “Upload_CKAN_with_decompresion” template and below you can find the instructions to run it.

In this case the user should also configure following processor (before Step 2 in the previous template):
Data acquisition framework

1. Step 1: Go to “InvokeHTTP” processor, go to “properties” tab and fulfill:
   - **Mandatory**:
     - “Remote URL”, fill with the remote URL form where to get the dataset.
   - **Others**. Apart from the mandatory properties, other properties exist in the processor and they can remain as they are or be configured by the user. Some of them are:
     - “Basic Authentication Username”, “Basic Authentication Password” can be used for downloading with required authentication.
     - In “Scheduling” tab the user can select a “Timer driven” strategy and schedule the download task (and therefore the entire workflow).

2. Step 2: Go to “UnpackContent” processor, go to “properties” tab and fulfill
   - **Mandatory**:
     - “Packaging format”, type of compressed package. By default the “mime.type” value can automatically detect and decompress the file.
     - “File filter”, the user should specify in which files is interested from the extracted ones. It will be setting by Indicating the extension(s) of the file(s) by means of a regular expression such as: [.gml|.xml] or [.gml|.xml|.zip]

3. Step 3: Go to “Define_Package_Name processor”, go to “properties” tab and fulfill:
   - **Mandatory**
     - “CKAN_package_name”, the name of the dataset in CKAN. Only accept letters, hyphen, and lowercase.

4. Step 4: Go to “CKAN_Package_Backup” processor, go to “properties” tab and fulfill:
   - **Mandatory**
     - “CKAN Url”, URL of the CKAN repository
     - “File Api_key”, API Key for operating with CKAN API.
     - “Name of the package to backup”, Name of the dataset to be backedumped. If the dataset does not exist, nothing happens. If the dataset exist, a copy of the specified dataset will be created. According to the normative defined in Section 3.3, just “fresh” datasets are subject to be backedumped. It should be lowercase.
     - “Comma-separated Tag List”, the user might specify different alphanumeric tags. As it was explained in Section 3.3 at least three tags should be fulfilled to backup a new dataset:
       - One tag indicating the project name: “String”. By default and in the context of the QROWD project the user must set always: “QROWD”
       - One tag indicating the type of dataset. As we are configuring the backup processor, the user must set always: “QROWD_historical”
       - One tag indicating the type of dataset (from a use point of view). In the context of QROWD the user has to select one of the following values:{QROWD_source, QROWD_fusion, QROWD_official}. 
5. Step 5: Go to “CKAN_Flowfile_Uploader” processor, go to “properties” tab and fulfill:
   - Mandatory
     - “CKAN Url”, URL of the CKAN repository
     - “File Api_key”, API Key for operating with CKAN API.
     - “Organization id”, the creator/owner of the dataset. This property is always needed, the first time to add the organization to a dataset, and following times to check if the organization passed corresponds to the ownership of dataset in CKAN. It should be lowercase.
     - “Name_of_the_package”, the CKAN dataset that will host the new version. According to the normative defined in Section 3.3, just “fresh” datasets are subject to keep the last version. It should be lowercase.
     - “Package visibility”, the accessibility of the dataset: “public”, if anyone can access to the dataset without restrictions, or “private” restricted to some users.
     - “Comma-separated Tag List”, here the user might specify different alphanumeric tags. As it was explained in Section 3.3 at least three tags should be fulfilled to upload a new dataset:
       - One tag indicating the project name: “String”. By default and in the context of the QROWD project the user must set always: “QROWD”.
       - One tag indicating the type of dataset. As we are configuring the uploader processor, the user must set: “QROWD_lastVersion”
       - One tag indicating the type of dataset (from a use point of view). In the context of QROWD the user has to select one of the following values: {QROWD_source, QROWD_fusion, QROWD_official}
         - Example: “QROWD, QROWD_lastVersion, QROWD_fusion”.

CSDA-3: JSON-based transformations + Update a dataset into CKAN

Figure 15 shows the NiFi GUI to add a new element to the NiFi dataflow based on previous templates. In this case the user should select “JSON_based_transformation_upload_CKAN” template and below you can find the instructions to run it.
Data acquisition framework

1. Step 1: Go to “InvokeHTTP” processor, go to “properties” tab and fulfill:
   - Mandatory:
     - “Remote URL”, fill with the remote URL form where to get the dataset.
   - Others. Apart from the mandatory properties, other properties exist in the processor and they can remain as they are or be configured by the user. Some of them are:
     - “Basic Authentication Username”, “Basic Authentication Password” can be used for downloading with required authentication.
     - In “Scheduling” tab the user can select a “Timer driven” strategy and schedule the download task (and therefore the entire workflow).

2. Step 2: Go to “SplitJson” processor, go to “properties” tab and fulfill:
   - Mandatory:
     - “JsonPath Expression”: Introduce a JSON path expression to point out how to split the input JSON. i.e: by setting “$.*” this processor will split the flow file (at this point, just a single one) into as many flow files such elements in the top level array in the JSON input. Make use of 15

3. Step 3: Go to “EvaluateJsonPath” processor, go to “properties” tab and add (button +) attributes to store JSON content from each flow file. This information can be needed for further processors.

15 http://jsonpath.com/
Data acquisition framework

- **Mandatory**
  - “id”. A JSON path expression that retrieves the id of the JSON object. Needed for the Merge.

- **Other**
  - add custom user attributes to flowfile attributes. The value for these attributes will be retrieved from the JSON object processed as flowfile. These attributes can be used in the “JoltTransformationJSON” processor.

4. **Step 4:** Go to “UpdateAttribute” processor, go to “properties” tab and fulfill:
   - **Other**
     - “type”, fill the “type”, a string to indicate a shared type for all the elements. It can be useful to indicate the FIWARE type when the transformation is from JSON structure to FIWARE structure.

5. **Step 5:** Go to “JoltTransformationJSON” processor, go to “Advance” (button left side) and mandatorily introduce the JOLT specification which will transform an input JSON structure into another JSON structure. A reference can be found here[^16]. A new flowfile is created with the new structure.

6. **Step 6:** “MergeContent” processor does not need to be parameterized. After this processor all the flowfiles will be merge into a single one. Internally the processor will make use of the attribute “id” defined in “EvaluateJsonPath”.

7. **Step 7:** Go to “UpdateAttribute” processor, go to “properties” tab and fulfill:
   - **Mandatory**
     - “filename”, the name of the flowfile. This name will be used by “CKAN_Flowfile_Uploader” to set the name of the resource (or file) to be added to CKAN. It is a fixed string.

8. **Step 8:** Go to “CKAN_Flowfile_Uploader” processor, go to “properties” tab and fulfill:
   - **Mandatory**
     - “CKAN Url”, URL of the CKAN repository
     - “File Api_key”, API Key for operating with CKAN API.
     - “Organization id”, the creator/owner of the dataset. This property is always needed, the first time to add the organization to a dataset, and following times to check if the organization passed corresponds to the ownership of dataset in CKAN. It should be lowercase.
     - “Name_of_the_package”, the CKAN dataset that will host the new format. According to the normative in Section 3.3, the output of the transformation (the new format) is assumed to be uploaded as a new resource (new file) into an existing dataset, in particular into the dataset from which the file to be transformed was downloaded. It should be lowercase.
     - “Package visibility”, the accessibility of the dataset: “public”, if anyone can access to the dataset without restrictions, or “private” restricted to some users.
     - “Comma-separated Tag List”, here the user might specify different alphanumeric tags. As it was explained in Section 3.3 at least three tags should be fulfilled to upload a new dataset:

[^16]: http://jolt-demo.appspot.com/#inception
Data acquisition framework

- One tag indicating the project name: “String”. By default and in the context of the QROWD project the user must set always: “QROWD”.
- One tag indicating the type of dataset. As we are configuring the uploader processor, the user must set: “QROWD_lastVersion”
- One tag indicating the type of dataset (from a use point of view). In the context of QROWD the user has to select one of the following values: {QROWD_source, QROWD_fusion, QROWD_official}
- Example: “QROWD, QROWD_lastVersion, QROWD_fusion”.

CSDA-4: External-script transformations + Update a dataset into CKAN

Figure 16 shows the NiFi GUI to add a new element to the NiFi dataflow based on previous templates. In this case the user should select “External_script_transformation_upload_CKAN” template and below you can find the instructions to run it.

![Figure 16: CSDA-4 NiFi template](image-url)
1. Step 1: Go to “InvokeHTTP” processor, go to “properties” tab and fulfill:
   - Mandatory:
     - “Remote URL”, fill with the remote URL form where to get the dataset.
   - Others. Apart from the mandatory properties, other properties exist in the processor and they can remain as they are or be configured by the user. Some of them are:
     - “Basic Authentication Username”, “Basic Authentication Password” can be used for downloading with required authentication.
     - In “Scheduling” tab the user can select a “Timer driven” strategy and schedule the download task (and therefore the entire workflow).

2. Step 2: Go to “UpdateAttribute” processor, go to “properties” tab and fulfill:
   - Mandatory:
     - “filename”, fill the name of the flowfile, that will be needed for the “PutFile” processor to set file in the directory. It is a fixed string.

3. Step 3: Go to “PutFile”, processor, go to “properties” tab and fulfill:
   - Mandatory
     - “directory”, set the directory where the flowfile payload will be downloaded in the form of a file. It will be the input file for the next processor.

4. Step 4: Go to “ExecuteStreamCommand” processor, go to “properties” tab and fulfill:
   - Mandatory:
     - “Command Path”, the needed command to run the script. As a matter of example: “ogr2ogr” a library which is included and is able to transform from any format into JSON.
     - “Command Argument”, the needed arguments to run the command. As a matter of example for the ogr2ogr library: -f, “output desired format”;“output file”;“input file”;-s_srs;EPSG:25832;-t_srs;EPSG:4326

5. Step 5: Go to “FetchFile” processor, go to “properties” tab and fulfill:
   - Mandatory:
     - “File To Fetch”, set the “output file”. Must be the same of the output file obtained in the previous processor.

6. Step 6: Go to “UpdateAttribute” processor, go to “properties” tab and fulfill:
   - Mandatory:
     - “filename”, the name of the flowfile. This name will be used by “CKAN_Flowfile_Uploader” to set the name of the resource (or file) to be added to CKAN. It is a fixed string.

7. Step 7: Go to “CKAN_Flowfile_Uploader” processor, go to “properties” tab and fulfill:
   - Mandatory
     - “CKAN Url”, URL of the CKAN repository
     - “File Api_key”, API Key for operating with CKAN API.
     - “Organization id”, the creator/owner of the dataset. This property is always needed, the first time to add the organization to a dataset, and following times to check if the organization passed
Data acquisition framework

corresponds to the ownership of dataset in CKAN. It should be lowercase.
- “Name_of_the_package”, the CKAN dataset that will host the new format. According to the normative in Section 3.3, the output of the transformation (the new format) is assumed to be uploaded as a new resource (new file) into an existing dataset, in particular into the dataset from which the file to be transformed was downloaded. It should be lowercase.
- “Package visibility”, the accessibility of the dataset: “public”, if anyone can access to the dataset without restrictions, or “private” restricted to some users.
- “Comma-separated Tag List”, here the user might specify different alphanumeric tags. As it was explained in Section 3.3 at least three tags should be fulfilled to upload a new dataset:
  - One tag indicating the project name: “String”. By default and in the context of the QROWD project the user must set always: “QROWD”.
  - One tag indicating the type of dataset. As we are configuring the uploader processor, the user must set: “QROWD_lastVersion”
  - One tag indicating the type of dataset (from a use point of view). In the context of QROWD the user has to select one of the following values: {QROWD_source, QROWD_fusion, QROWD_official}
  - Example: “QROWD, QROWD_lastVersion, QROWD_fusion”.

3.6. Application of static core templates: Integrated acquisition and FIWARE/GeoJSON transformation of Municipality of Trento datasets

The static data acquisition framework provides a set of core NiFi templates that can be used and connected between them in order to create more complex workflows. As part of the framework, and as an example of utilization of core templates, a set of more advanced ad-hoc and parameterized workflows will be put available for the project.

These workflows will be specifically focused in a integrated acquisition and transformations (FIWARE, GeoJSON) of Municipality of Trento datasets required to be visualized in the dashboard. In particular, each workflow will be composed of following sub-workflows based on the core templates:

- Sub-workflow 1: Original datasets uploaded to CKAN.
- Sub-workflow 2: Original datasets transformed into GeoJSON format
- Sub-workflow 3: GeoJSON dataset dataset transformed into FIWARE dataset
For the creation of these workflows it is assumed some basic NiFi knowledge, since it is necessary to add some NiFi components (ports, connections, etc.) to/between the existing templates.

Figure 18 shows an example of the workflow. In particular the workflow is for the acquisition of “impianti-sportivi” dataset.
Data acquisition framework

● sub-workflow “impianti-sportivi Acquisition”, it is built from CSDA-1 template and aims to:
  ○ download the “Sport facilities” dataset from the Municipality services\(^{17}\) and
  ○ upload it into CKAN in the resource “impianti_sportivi.gml” of a dataset named “impianti-sportivi-mt”.
● sub-workflow “impianti-sportivi GeoJSON Transformation”, it is built from CSDA-4 template and aims to:
  ○ download the “impianti_sportivi.gml” format from the dataset “impianti-sportivi-mt”,
  ○ transform it into GeoJSON format, and
  ○ upload the new format as new resource “impianti_sportivi_GEOJSON.json” in the “impianti-sportivi-mt” dataset.
● sub-workflow “impianti-sportivi FIWARE Transformation”, it is built from CSDA-3 template and aims to:
  ○ download the “impianti_sportivi_GEOJSON.json” format from the dataset “impianti-sportivi-mt”,
  ○ transform it into a FIWARE format and
  ○ upload the new format as new resource “impianti_sportivi_FIWARE.json” in the “impianti-sportivi-mt” dataset.

Each of the sub-workflow now incorporates (input or/and output) ports, as new NiFi component, to enable the connection between them.

3.7. **Deployment and release of the framework**

**Deployment**

The following NiFi components has been deployed into the QROWD server in Leipzig hosted by InfAI with partner number 7:

- **Basic Templates**
  - CSDA-1: Upload/Update a dataset into CKAN (from a URL)
  - CSDA-2: Upload/Update a dataset into CKAN (from a URL + decompress)
  - CSDA-3: JSON-based transformations + Update a dataset into CKAN
  - CSDA-4: External-script transformations + Update a dataset into CKAN
- **N advanced workflows. Please refer to table in ANNEX 1: Deployed dataflows to see a relation of dataflows deployed.**
- **Bundles**

**Release**

The CSDA templates and the specific new NiFi processors developed are released under licence Apache 2.0 on the following QROWD git repositories:

- CSDA templates\(^\text{18}\), collection of templates created for the QROWD project to be used in NiFi.
- NiFi CKAN processors\(^\text{19}\), custom Apache Nifi processor to upload files to CKAN.

3.8. **Static Requirements Validation**

D4.1 defined a set of requirements that the ACSD framework should accomplish. This section aims at assessing the degree of fulfillment of those requirements, as depicted in Table 1.

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement name</th>
<th>Description</th>
<th>Notes</th>
<th>Validation</th>
<th>Notes V</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA-101</td>
<td>Dataset consolidation</td>
<td>The ACSD should collect datasets from several data sources</td>
<td>i.e: Able to gets datasets from several portals, from open services...</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>DA-102</td>
<td>Dataset access type</td>
<td>The ACSD should handle different ways of accessing to the datasets</td>
<td>i.e: Retrieve data directly from files or from web API,...</td>
<td>PA</td>
<td>Any resource accessible from a remote URL</td>
</tr>
<tr>
<td>DA-103</td>
<td>Dataset</td>
<td>The ACSD should</td>
<td>i.e: An</td>
<td>A</td>
<td>CKAN</td>
</tr>
</tbody>
</table>

\(^{18}\) https://github.com/QROWD/NiFi-templates
\(^{19}\) https://github.com/QROWD/nifiCkanProcessor
<table>
<thead>
<tr>
<th>DA-104</th>
<th>Metadata about data</th>
<th>Information or &quot;metadata&quot; about the data</th>
<th>i.e: Information about the data: title, publisher, date, schema file, provenance. ..</th>
<th>A</th>
<th>Each dataset will have basic tags for owner (publisher), visibility, and a associated metadata file for provenance metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA-105</td>
<td>Access mechanisms</td>
<td>The ACSD should provide means to allow subsequent systems to browse and find the data</td>
<td>i.e: RDF-transformation component should be able to access the data</td>
<td>PA</td>
<td>The Backend consumer system will allow retrieve datasets based on its identifier</td>
</tr>
<tr>
<td>DA-106</td>
<td>Batch job execution</td>
<td>Ideally, the ACSD should pull data from the source systems at scheduled intervals using batch components.</td>
<td>i.e: Process executed as a batch job</td>
<td>A</td>
<td>The entry point NiFi processor (InvokeHTTP) of each acquisition template allow scheduling</td>
</tr>
<tr>
<td>DA-107</td>
<td>Definition of datasets acquisition process</td>
<td>A set of first steps toward starting up the acquisition process should be defined</td>
<td>i.e: For each new dataset define the process to import it including: access way, transformations, metadata, timeliness, etc...</td>
<td>A</td>
<td>For each acquisition template it is provided a guided with a detailed set of steps to be followed</td>
</tr>
<tr>
<td>DA-108</td>
<td>Versioning</td>
<td>The ACSD should be able to track different versions of datasets</td>
<td>i.e: Update datasets with new</td>
<td>A</td>
<td>A versioning mechanism is provided</td>
</tr>
</tbody>
</table>
Data acquisition framework

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement name</th>
<th>Description</th>
<th>Notes</th>
<th>Validation</th>
<th>Notes V</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA-009</td>
<td>Security restrictions</td>
<td>Datasets can be public or private</td>
<td>i.e: Some of the used datasets are the private usage</td>
<td>A</td>
<td>Dataset in CKAN can be public or private</td>
</tr>
<tr>
<td>DA-110</td>
<td>ETL process</td>
<td>For some datasets the ACSD could implement classical ETL process</td>
<td>i.e: A simple extraction, transformation and loading process could be applied to specific datasets</td>
<td>A</td>
<td>A &quot;JSON-based transformation and upload CKAN&quot; template is provided to allow basic extract, transformation and upload process</td>
</tr>
<tr>
<td>DA-111</td>
<td>File format access type</td>
<td>The ACSD should be able to access/transform data from several file format</td>
<td>i.e: XML, JSON, GML</td>
<td>A</td>
<td>The &quot;External-script transformation and upload CKAN&quot; allow transformations between different formats</td>
</tr>
</tbody>
</table>

Non-functional requirements

Table 2: Static requirements validation (NFR)

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement name</th>
<th>Description</th>
<th>Notes</th>
<th>Validation</th>
<th>Notes V</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA-112</td>
<td>Flexibility</td>
<td>The ACSD should be able to incorporate unknown, new or changing datasets.</td>
<td>i.e: The load of a new dataset should be carried with the less possible number of changes</td>
<td>A</td>
<td>Each of the acquisition templates allows the incorporation of new dataset just by adjusting parameters.</td>
</tr>
</tbody>
</table>
Based on this, we can state that the majority of the requirements of the ACSD have been accomplished.

4. DYNAMIC DATA ACQUISITION

4.1. Datasets

The framework described in following sections provides ways to acquire datasets of dynamic or streaming nature.

We defined real-time or streaming data for the purposes of QROWD in D4.1 as, “a type of dynamic data with a very high or continuously rate of change and usually with the need to be consumed immediately after its production”. Therefore, in the case of the pilots of QROWD, a streaming dataset can be seen as collections of data that provide information about the current status of the city, such as real-time availability of bike-sharing, real time status of underground parking, etc.

Deliverable D4.1 presented a catalog of datasets of real-time or streaming nature to be used in QROWD pilots. The data acquisition framework intends to give support to that type of datasets.

4.2. Data flow

Figure 19 shows the main elements needed to acquire, update and transform dynamic datasets (i.e. Underground parking status, Bike-racks for bike-sharing status) in the FIWARE Orion Context Broker using NiFi.

![Figure 19: QROWD dynamic data acquisition framework](image-url)
In the case of dynamic acquisition, a single process will be in charge of:
- retrieving real time information for available services,
- transforming the original data into FIWARE entities,
- and uploading and updating entities into the Context Broker through the well-defined FIWARE NGSI API\(^{20}\).

The entire acquisition process is made through a set of NiFi-based processors which make easier the integration with further components of the QROWD general architecture (Figure 1).

### 4.3. Context Broker general considerations

The main consideration the user should take into account to operate with the component provided is that all the entities posted or updated into the Context Broker should fit the FIWARE data models\(^{21}\) where possible. For instance, FIWARE provides data models for parking, such as off-street parking\(^{22}\), that are of interest to represent specific datasets managed by QROWD.

### 4.4. Core dynamic data acquisition functionality

#### 4.4.1. CDDA-5: JSON based transformation + Update to Context broker

This NiFi process provides the means to take a given dataset in JSON format and transform it to the FIWARE model and update it to the Context Broker as shown in Figure 20.

---

21 https://www.fiware.org/developers/data-models/
The dataflow shown in Figure 20 consists of two main processor groups::

- “ConvertToFiware”, group of processors in charge of transforming a JSON format into FIWARE data model. This processor group flow is shown in Figure 21.
- “Copy of FiwareRESTAPIHandler”, a group of components in charge of uploading entities to FIWARE context broker. This processor group flow is shown in Figure 22.
The logic of the dataflow shown in Figure 21 is quite similar to the initial steps presented in the static template “JSON-based transformations + Update a dataset into CKAN” in Sections 3.4 and 3.5. Refer back to those sections for further details.
The detail of the dataflow shown in Figure 22 is the following:

- “Evaluate JSON path” (estandar), processor in charge of getting the id of the entity.
- “Invoke HTTP” (estandar), processor in charge of checking if already exist an entity in the Context Broker with this id. If the answer is:
  - “Yes”, the following processors will be in charge of updating the existing entity in the context broker:
    - “RemoveIDForUpdate” (estandar), processor in charge of removing the “id” from the entity,
    - “PostUpdateToFWARE” (estandar), processor in charge of updating an existing FIWARE entity (sent in the request body) into the Context Broker by means of NGSI API. The id is sent as path param.
- “No”, the following processors will be in charge of posting a new entity in the context broker:
  - “JoltTransformation” (estandar), processor in charge of adding the FIWARE type to the entity.
  - “POSTNewEntityToFiware” (estandar), processor in charge of uploading a new FIWARE entity (sent in the request body) into the context broker by means of NGSI API. The id is sent in the entity itself.

4.5. **Guided Procedure**

![Add Template](image)

**Figure 23: CDDA-1 NiFi template**

2. Step 1: Go to “ConvertToFiware” group and follow same 5 first steps than section CSDA-3: JSON-based transformations + Update a dataset into CKAN.
3. Step 2: Go to “Copy of FiwareRESTAPIHandler” group and by double clicking fill the following steps:
   - Step 3: Go to “EvaluateJsonPath” processor, go to “properties” tab fulfill:
     - Mandatory
     - “id”. A JSON path expression that retrieve the id of the new context broker entity. Needed for checking if the entity already exist in the context broker.
   - In case of an update:
     - Step 4: Go to “Invoke HTTP” processor, go to “properties” tab and fulfill:
       - Mandatory:
         - In “Remote URL” property, replace Context_Broker in bold: `<Context_Broker>/v2/entities/${ID}` with the IP where the Context Broker is running.
Data acquisition framework

- Step 5: Go to “RemoveIDForUpdate” processor, go to “properties” tab and fulfill:
  - Mandatory
    - “Search Value”, fill in it with the regular expression that remove the id from the entity.
- Step 6: Go to “PostUpdateToFirmware” processor, go to “properties” tab and fulfill:
  - Mandatory
    - In “URL” property, replace Context_Broker in bold in \(<\text{Context}_{\text{broker}}>/v2/entities/${ID}/attrs?options=keyValues\) with the IP where the Context Broker is running.
- In case of a new upload:
  - Step 7: Go to “JoltTransformationJSON” processor, go to “Advance” (button left side) and mandatorily introduce the JOLT specification that will transform an input JSON structure into another JSON structure. A reference can be find here\(^{23}\). A new flowfile is created with the new structure.
  - Step 8: Go to “PostNewEntityToFirmware” processor, go to “properties” tab and fulfill:
    - Mandatory
      - In “URL” property, replace Context_Broker in bold in \(<\text{Context}_{\text{broker}}>/v2/entities?options=keyValues\) with the IP where the Context Broker is running.

4.6. Deployment of the framework

Deployment

Following components will be deployed into the QROWD servers:

- Basic Templates
- CDDA-5: JSON-based transformations + Update to Context broker
- Two applications of the previous template
  - Bike-Sharing acquisition dataflow
  - Underground parking availability

Release

The CSDA templates developed are released under licence Apache 2.0 on the following QROWD git repository:

- CSDA templates\(^{24}\), collection of templates created for the QROWD project to be used in NiFi.

---

23 http://jolt-demo.appspot.com/#inception
24 https://github.com/QROWD/NiFi-templates
4.7. Dynamic Requirements Validation

D4.1 defined a set of requirements to accomplish by the Dynamic acquisition framework. Table 3 and Table 4 show the assessment of the requirements’ coverage.

Functional requirements

Table 3: Dynamic requirements validation (FR)

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement name</th>
<th>Description</th>
<th>Notes</th>
<th>Validation</th>
<th>Notes V</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA-201</td>
<td>Acquisition of last measured data</td>
<td>Transfer the latest state of dynamic data from external data sources to a central broker/repository.</td>
<td>i.e: Put available to QROWD platform last state of weather forecast entities obtained from the Open Data Trentino portal</td>
<td>A</td>
<td>The CDDA-5 template allow to retrieve information in real time and store the last status in the Context Broker</td>
</tr>
<tr>
<td>DA-202</td>
<td>Currency factor</td>
<td>The acquisition tool(s) should be able to provide the desired latency in updating the data</td>
<td>i.e: The system should update and put available weather forecast information for Trento each 15 minutes.</td>
<td>A</td>
<td>The entry point NiFi processor (InvokeHTTP) of CDDA-5 allows scheduling (different latency in updating the dataset)</td>
</tr>
<tr>
<td>DA-203</td>
<td>Timeliness factor</td>
<td>Different datasets may have different rate of change.</td>
<td>i.e: every fifteen minute, hourly, 2 per day, daily, 15 days, quarterly, yearly.</td>
<td>A</td>
<td>The entry point NiFi processor (InvokeHTTP) of CDDA-5 allows selecting different elapsed amount of time to execute the updates</td>
</tr>
<tr>
<td>DA-204</td>
<td>File format access type</td>
<td>The system should be able to acquire data in different file format or access</td>
<td>i.e: XML, CSV-GTF, CSV, GML/SHP/KML/D</td>
<td>NA</td>
<td>Eventually there was not need in the</td>
</tr>
</tbody>
</table>
Data acquisition framework

schemes. Such as flat files, database dumps or SQL interfaces. The list of access types is driven by the characteristics of datasets. XF, JSON, SQL dumps, SQL interfaces, etc.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Data sources</th>
<th>Description</th>
<th>Notes</th>
<th>Validation</th>
<th>Notes V</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA-205</td>
<td>Variety of data sources</td>
<td>The system should be able quickly and easily to integrate and expose data from a variety of data sources using APIs.</td>
<td>A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Non-functional requirement

Table 4: Dynamic requirements validation (NFR)

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement name</th>
<th>Description</th>
<th>Notes</th>
<th>Validation</th>
<th>Notes V</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA-206</td>
<td>OASC compliant</td>
<td>The system, or part of it, should be compliant with NGSI API standard and FIWARE data models</td>
<td>i.e: NGSI API, FIWARE data model</td>
<td>A</td>
<td>The CDDA-5 template is compliant with the NFSI API</td>
</tr>
<tr>
<td>DA-207</td>
<td>Scalability</td>
<td>Building a scalable infrastructure able to handle huge datasets used in QROWD.</td>
<td>i.e: Import a dump file of 19,5 gb</td>
<td>A</td>
<td>The acquisition dynamic template is built with NiFi25, a scalable technology for scalable data flows</td>
</tr>
<tr>
<td>DA-208</td>
<td>Flexibility</td>
<td>The system should be able to incorporate unknown, new or changing datasets.</td>
<td>i.e: The load of a new dataset should be carried with the less possible number of changes</td>
<td>A</td>
<td>Each of the acquisition templates allows the incorporation of new dataset just by adjusting parameters</td>
</tr>
</tbody>
</table>

25 https://nifi.apache.org/
5. CONCLUSIONS

The document presented the main building blocks and functionalities offered by the QROWN Data Acquisition Framework. The objective of the document was to present the work done in the scope of WP4 for data acquisition, including software and methodological support to enable data ingestion and transformation to fulfil the requirements of the pilots of the project.

The users of the framework are mainly QROWN developers, but also any developer of data-enabled applications who would like to make ingest data into CKAN or the Orion Context Broker. Therefore the framework could be used to enable the acquisition of data in the scope of QROWN, but also in an isolated fashion enable data acquisition in the abovementioned repositories.

The main results presented in the document enable the acquisition of both static and dynamic datasets. The document describes a set data flows created by combination of Apache NiFi templates generated in the scope of QROWN. The templates can be used and combined by developers in the NiFi GUI to define actual dataflows to allow complex data acquisition pipelines to ingest static data into CKAN or dynamic data into the FIWARE Orion Context Broker with none or minimal programming effort.

The document provides hints and describes best practices to facilitate the creation of the data flows and perform simple data transformations in NiFi. The document shows how this is done to generate datasets in multiple formats and flavours and upload them to CKAN in the case of static data, or to the Context Broker in the case of dynamic data. A specific dataset versioning mechanism in CKAN has been implemented to enable the functionality needed in QROWN.

The document is accompanied with the deployment of the software of the Data Acquisition Framework in the QROWN server located at the InfAI premises in Leipzig. Most of the code will be made available under an open source license.
6. REFERENCES

Apache NiFi. Website, Available at https://nifi.apache.org/. Accessed October 31, 2018

CKAN. Website, Available at https://ckan.org Accessed October 31, 2018


FIWARE Data Models. Website, Available at https://www.fiware.org/developers/data-models/ Accessed October 31, 2018


OASC. Website, Available at http://oascities.org/. Accessed October 31, 2018

OpenStreetMap. Website, Available at https://www.openstreetmap.org/. Accessed October 31, 2018

7. **ANNEX 1: DEPLOYED DATAFLOWS**

Following table shows the data flows deployed in QROWN NiFi server and related information.

<table>
<thead>
<tr>
<th>Data flow</th>
<th>Dataset</th>
<th>Template</th>
<th>Description</th>
<th>File</th>
<th>Original URL</th>
<th>CKAN URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>Description</td>
<td>Data Sources</td>
<td>Web Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------</td>
<td>---------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1027</td>
<td>Bike sharing stations</td>
<td>CSDA-2, CSDA-4, CSDA-3</td>
<td><a href="http://webapps.comune.trento.it/cartografia/gis/dbexport?db=base&amp;sc=mobilita&amp;ly=centro_in_bici&amp;fr=gml">http://webapps.comune.trento.it/cartografia/gis/dbexport?db=base&amp;sc=mobilita&amp;ly=centro_in_bici&amp;fr=gml</a></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1028</td>
<td>Nursery</td>
<td>CSDA-2</td>
<td><a href="http://webapps.comune.trento.it/cartografia/gis/dbexport?db=base&amp;sc=mobilita&amp;ly=nidi&amp;fr=gml">http://webapps.comune.trento.it/cartografia/gis/dbexport?db=base&amp;sc=mobilita&amp;ly=nidi&amp;fr=gml</a></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data acquisition framework
<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>CSDA Levels</th>
<th>Source</th>
<th>Data URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>Description</td>
<td>Type</td>
<td>Source</td>
<td>Web Link</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------</td>
<td>-----------------------------</td>
<td>----------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1032</td>
<td>biblioteche</td>
<td>Municipal libraries</td>
<td>biblioteche-mt</td>
<td><a href="http://webapps.comune.trento.it/cartografia/gis/dbexport?db=base&amp;sc=istruzioni_sport&amp;ly=biblioteche&amp;fr=gml">http://webapps.comune.trento.it/cartografia/gis/dbexport?db=base&amp;sc=istruzioni_sport&amp;ly=biblioteche&amp;fr=gml</a></td>
</tr>
<tr>
<td>1041</td>
<td>taxi_stations</td>
<td>Taxi stations</td>
<td>taxi-stations-mt</td>
<td><a href="http://ckan.growd.aksw.org/dataset/taxi-stations-mt">http://ckan.growd.aksw.org/dataset/taxi-stations-mt</a></td>
</tr>
<tr>
<td>1052</td>
<td>parking</td>
<td>Parking</td>
<td>parking-</td>
<td>http://</td>
</tr>
<tr>
<td>parking_meter</td>
<td>meters of Trento</td>
<td>meter-mt</td>
<td>ckan.growd.aks w.org/dataset/e0b8e0df-cf14-466f-b424-60b46487930f/resource/c8d92284-b859-408d-83cc-8425f28fe8cf/download/parking-meter-mt_geojson.geojson</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>----------</td>
<td>------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>1058 bikeracks</td>
<td>1058 CSDA-2, CSDA-4, CSDA-3</td>
<td>Bike racks in LTZ area</td>
<td>bikeracks-mt</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><a href="http://ckan.growd.aks">http://ckan.growd.aks</a> w.org/dataset/bikeracks-mt</td>
<td></td>
</tr>
<tr>
<td>1060 superiori</td>
<td>1060 CSDA-2, CSDA-4, CSDA-3</td>
<td>High schools</td>
<td>superiori-mt</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><a href="http://ckan.growd.aks">http://ckan.growd.aks</a> w.org/dataset/superiori-mt</td>
<td></td>
</tr>
<tr>
<td>1061</td>
<td>1061 CSDA-2, Carsharing</td>
<td>car-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data acquisition framework</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>car_sharing</strong></td>
<td>CSDA-4, CSDA-3</td>
<td>Trentino parking slots</td>
<td>sharing-mt</td>
<td>Acquisition from Trento Municipality, GeoJSON transformation, Fiware.</td>
</tr>
<tr>
<td>Pollution</td>
<td>Pollution</td>
<td>CSDA-1</td>
<td>Air quality</td>
<td>airquality-mt</td>
</tr>
</tbody>
</table>

**New datasets added in v0.2 of the deliverable**

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>IDV</th>
<th>Source</th>
<th>URL</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1056</td>
<td>Neighborhoods _valley_Trento</td>
<td>1056</td>
<td>CSDA-3</td>
<td><a href="http://ckan.qrowd.aks">http://ckan.qrowd.aks</a> w.org/dataset/a2c95287-fcf8-47cf-8797-ada82580f8b2/resource/dc4ca58a-303d-45a7-9c63-ab1842b0dda0/download/neighbourhoods utmlatlon.geojson</td>
<td><a href="http://ckan.qrowd.aks">http://ckan.qrowd.aks</a> w.org/dataset/neighbourhoods-trento-mt</td>
</tr>
<tr>
<td>Dataset Name</td>
<td>Dataset ID</td>
<td>Dataset Type</td>
<td>Description</td>
<td>JSON Location</td>
<td>Dataset Location</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------</td>
<td>--------------</td>
<td>----------------------</td>
<td>--------------------------------------------------------------------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>4001_Bike_Racks_VCE</td>
<td>4001</td>
<td>CSDA-1</td>
<td>Bike racks vce</td>
<td><a href="http://waisvm-rcg1v07.ecs.soton.ac.uk/vce/4001_Bike_Racks_VCE_Geo">http://waisvm-rcg1v07.ecs.soton.ac.uk/vce/4001_Bike_Racks_VCE_Geo</a> JSON.json</td>
<td><a href="http://ckan.qrowd.aksw.org/dataset/bike-racks-vce">http://ckan.qrowd.aksw.org/dataset/bike-racks-vce</a></td>
</tr>
<tr>
<td>4002_Bike_Racks_OSM</td>
<td>4002</td>
<td>CSDA-1</td>
<td>Bike racks osm</td>
<td><a href="http://waisvm-rcg1v07.ecs.soton.ac.uk/vce/4002_Bike_Racks_OSM_Geo">http://waisvm-rcg1v07.ecs.soton.ac.uk/vce/4002_Bike_Racks_OSM_Geo</a> JSON.json</td>
<td><a href="http://ckan.qrowd.aksw.org/dataset/bike-racks-osm">http://ckan.qrowd.aksw.org/dataset/bike-racks-osm</a></td>
</tr>
<tr>
<td>TrentoNord Sensor</td>
<td>A22-Nord</td>
<td>CSDA-3</td>
<td>Trento Sensors A22 Nord</td>
<td><a href="https://api-test.smartcommunitylab.it/trento.mobilitiesensordata/1.0.0/traffic/RDT/ByHour">https://api-test.smartcommunitylab.it/trento.mobilitiesensordata/1.0.0/traffic/RDT/ByHour</a></td>
<td><a href="http://ckan.qrowd.aksw.org/dataset/trentonord-mt">http://ckan.qrowd.aksw.org/dataset/trentonord-mt</a></td>
</tr>
<tr>
<td>TrentoSud Sensor</td>
<td>A22-Sud</td>
<td>CSDA-3</td>
<td>Trento Sensors A22 Sud</td>
<td><a href="https://api-test.smartcommunitylab.it/trento.mobilitiesensordata/1.0.0/traffic/RDT/ByHour">https://api-test.smartcommunitylab.it/trento.mobilitiesensordata/1.0.0/traffic/RDT/ByHour</a></td>
<td><a href="http://ckan.qrowd.aksw.org/dataset/trentosud-mt">http://ckan.qrowd.aksw.org/dataset/trentosud-mt</a></td>
</tr>
<tr>
<td>TrentoSS47 Sensor</td>
<td>SS47</td>
<td>CSDA-3</td>
<td>Trento Sensors SS47</td>
<td><a href="https://api-test.smartcommunitylab.it/trento.mobilitiesensordata/1.0.0/traffic/RDT/ByHour">https://api-test.smartcommunitylab.it/trento.mobilitiesensordata/1.0.0/traffic/RDT/ByHour</a></td>
<td><a href="http://ckan.qrowd.aksw.org/dataset/trentoss47-mt">http://ckan.qrowd.aksw.org/dataset/trentoss47-mt</a></td>
</tr>
<tr>
<td>TrentoPiedi Sensor</td>
<td>Piedicastello</td>
<td>CSDA-3</td>
<td>Trento Sensors Piedicastello</td>
<td><a href="https://api-test.smartcommunitylab.it/trento.mobilitiesensordata/1.0.0/traffic/RDT/ByHour">https://api-test.smartcommunitylab.it/trento.mobilitiesensordata/1.0.0/traffic/RDT/ByHour</a></td>
<td><a href="http://ckan.qrowd.aksw.org/dataset/trentopiedi-mt">http://ckan.qrowd.aksw.org/dataset/trentopiedi-mt</a></td>
</tr>
<tr>
<td>TrentoBikeSud</td>
<td>Bike Sud</td>
<td>CSDA-3</td>
<td>Trento</td>
<td><a href="https://api-test.smartcommunitylab.it/trento.mobilitiesensordata/1.0.0/traffic/RDT/ByHour">https://api-test.smartcommunitylab.it/trento.mobilitiesensordata/1.0.0/traffic/RDT/ByHour</a></td>
<td>http://</td>
</tr>
</tbody>
</table>
## Data acquisition framework

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Bike</th>
<th>CSDA-3</th>
<th>Trento Sensors Bike Nord</th>
<th>TrentoBikeNord Sensor</th>
<th>ud-mt</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sensors Bike Sud</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>test.smartcom</strong> <strong>munitylab.it/</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>trento.mobilitys</strong> <strong>ensordata/</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>1.0.0/traffic/</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Narx/ByHour</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>ckan.qrowd.aks</strong> <strong>w.org/dataset/</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>trentobikesud-mt</strong></td>
</tr>
<tr>
<td><strong>TrentoBikeNord Sensor</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong><a href="http://ckan.qrowd.aks">http://ckan.qrowd.aks</a></strong> <strong>w.org/dataset/</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>trentobikenord-mt</strong></td>
</tr>
</tbody>
</table>